GBDT算法原理与系统设计简介

wepon
http://wepon.me
2017.09.20

内容

- 泰勒公式
- 最优化方法
  - 梯度下降法（Gradient descend method）
  - 牛顿法（Newton’s method）
- 从参数空间到函数空间
  - 从Gradient descend 到 Gradient boosting
  - 从Newton’s method 到 Newton Boosting
- Gradient Boosting Tree 算法原理
- Newton Boosting Tree 算法原理：详解 XGBoost
- 更高效的工具包 LightGBM
- 参考文献

泰勒公式

- **定义：**泰勒公式是一个用函数在某点的信息描述其附近取值的公式。局部有效性

泰勒公式

- **定义：**泰勒公式是一个用函数在某点的信息描述其附近取值的公式。局部有效性

- **基本形式：** $$f(x)=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$$

  - 一阶泰勒展开： $f(x)\approx f(x_0)+f'(x_0)(x-x_0)$

  - 二阶泰勒展开： $f(x)\approx f(x_0)+f'(x_0)(x-x_0)+f''(x_0)\frac{(x-x_0)^2}{2}$

泰勒公式

- **定义：**泰勒公式是一个用函数在某点的信息描述其附近取值的公式。局部有效性
- **基本形式：**
$$f(x)=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$$
  - 一阶泰勒展开：$f(x)\approx f(x_0)+f'(x_0)(x-x_0)$
  - 二阶泰勒展开：$f(x)\approx f(x_0)+f'(x_0)(x-x_0)+f''(x_0)\frac{(x-x_0)^2}{2}$
- **迭代形式：**假设 $x^t=x^{t-1}+\Delta x$ ，将 $f(x^t)$ 在 x^{t-1} 处进行泰勒展开

$$f(x^t)=f(x^{t-1}+\Delta x)$$
$$\approx f(x^{t-1})+f'(x^{t-1})\Delta x+f''(x^{t-1})\frac{\Delta x^2}{2}$$

梯度下降法（Gradient Descend Method）

在机器学习任务中，需要最小化损失函数 $L(\theta)$，其中 θ 是要求解的模型参数。梯度下降法常用来求解这种无约束最优化问题，它是一种迭代方法：选取初值 θ^0，不断迭代，更新 θ 的值，进行损失函数的极小化。

梯度下降法（Gradient Descend Method）

在机器学习任务中，需要最小化损失函数 $L(\theta)$，其中 θ 是要求解的模型参数。梯度下降法常用来求解这种无约束最优化问题，它是一种迭代方法：选取初值 θ^0，不断迭代，更新 θ 的值，进行损失函数的极小化。

- 迭代公式： $\theta^t = \theta^{t-1} + \Delta\theta$

梯度下降法（Gradient Descend Method）

在机器学习任务中，需要最小化损失函数 $L(\theta)$，其中 θ 是要求解的模型参数。梯度下降法常用来求解这种无约束最优化问题，它是一种迭代方法：选取初值 θ^0，不断迭代，更新 θ 的值，进行损失函数的极小化。

- 迭代公式： $\theta^t = \theta^{t-1} + \Delta\theta$
- 将 $L(\theta^t)$ 在 θ^{t-1} 处进行一阶泰勒展开：

$$
\begin{aligned}
L(\theta^t) &= L(\theta^{t-1} + \Delta\theta) \\
&\approx L(\theta^{t-1}) + L'(\theta^{t-1})\Delta\theta
\end{aligned}
$$

梯度下降法（Gradient Descend Method）

在机器学习任务中，需要最小化损失函数 $L(\theta)$，其中 θ 是要求解的模型参数。梯度下降法常用来求解这种无约束最优化问题，它是一种迭代方法：选取初值 θ^0，不断迭代，更新 θ 的值，进行损失函数的极小化。

- 迭代公式： $\theta^t = \theta^{t-1} + \Delta\theta$
- 将 $L(\theta^t)$ 在 θ^{t-1} 处进行一阶泰勒展开：

$$L(\theta^t) = L(\theta^{t-1} + \Delta\theta)$$
$$\approx L(\theta^{t-1}) + L'(\theta^{t-1})\Delta\theta$$

- 要使得 $L(\theta^t) < L(\theta^{t-1})$，可取：$\Delta\theta = -\alpha L'(\theta^{t-1})$，则：$\theta^t = \theta^{t-1} - \alpha L'(\theta^{t-1})$

梯度下降法（Gradient Descend Method）

在机器学习任务中，需要最小化损失函数 $L(\theta)$，其中 θ 是要求解的模型参数。梯度下降法常用来求解这种无约束最优化问题，它是一种迭代方法：选取初值 θ^0，不断迭代，更新 θ 的值，进行损失函数的极小化。

- 迭代公式： $\theta^t = \theta^{t-1} + \Delta\theta$
- 将 $L(\theta^t)$ 在 θ^{t-1} 处进行一阶泰勒展开：

$$L(\theta^t) = L(\theta^{t-1} + \Delta\theta)$$
$$\approx L(\theta^{t-1}) + L'(\theta^{t-1})\Delta\theta$$

- 要使得 $L(\theta^t) < L(\theta^{t-1})$，可取：$\Delta\theta = -\alpha L'(\theta^{t-1})$，则：$\theta^t = \theta^{t-1} - \alpha L'(\theta^{t-1})$

这里 α 是 步长，可通过line search确定，但一般直接赋一个小的数。

牛顿法（Newton's Method）

- 将 $L\left(\theta^{t}\right)$ 在 θ^{t-1} 处进行二阶泰勒展开：

$$L\left(\theta^{t}\right) \approx L\left(\theta^{t-1}\right) + L^{'}\left(\theta^{t-1}\right)\Delta\theta + L^{''}\left(\theta^{t-1}\right)\frac{\Delta\theta^{2}}{2}$$

牛顿法（Newton's Method）

- 将 $L\left(\theta^{t}\right)$ 在 θ^{t-1} 处进行二阶泰勒展开：

$$L\left(\theta^{t}\right) \approx L\left(\theta^{t-1}\right) + L^{'}\left(\theta^{t-1}\right)\Delta\theta + L^{''}\left(\theta^{t-1}\right)\frac{\Delta\theta^{2}}{2}$$

为了简化分析过程，假设参数是标量（即 θ 只有一维），则可将一阶和二阶导数分别记为 g 和 h：

$$L\left(\theta^{t}\right) \approx L\left(\theta^{t-1}\right) + g\Delta\theta + h\frac{\Delta\theta^{2}}{2}$$

牛顿法（Newton's Method）

- 将 $L\left(\theta^t\right)$ 在 θ^{t-1} 处进行二阶泰勒展开：

$$L\left(\theta^t\right) \approx L\left(\theta^{t-1}\right) + L^{'}\left(\theta^{t-1}\right)\Delta\theta + L^{''}\left(\theta^{t-1}\right)\frac{\Delta\theta^2}{2}$$

为了简化分析过程，假设参数是标量（即 θ 只有一维），则可将一阶和二阶导数分别记为 g 和 h：

$$L\left(\theta^t\right) \approx L\left(\theta^{t-1}\right) + g\Delta\theta + h\frac{\Delta\theta^2}{2}$$

- 要使得 $L\left(\theta^t\right)$ 极小，即让 $g\Delta\theta + h\frac{\Delta\theta^2}{2}$ 极小，可令：

$$\frac{\partial\left(g\Delta\theta + h\frac{\Delta\theta^2}{2}\right)}{\partial\Delta\theta} = 0$$

求得 $\Delta\theta = -\frac{g}{h}$，故 $\theta^t = \theta^{t-1} + \Delta\theta = \theta^{t-1} - \frac{g}{h}$

牛顿法（Newton's Method）

- 将 $L\left(\theta^t\right)$ 在 θ^{t-1} 处进行二阶泰勒展开：

$$L\left(\theta^t\right) \approx L\left(\theta^{t-1}\right) + L^{'}\left(\theta^{t-1}\right)\Delta\theta + L^{''}\left(\theta^{t-1}\right)\frac{\Delta\theta^2}{2}$$

为了简化分析过程，假设参数是标量（即 θ 只有一维），则可将一阶和二阶导数分别记为 g 和 h：

$$L\left(\theta^t\right) \approx L\left(\theta^{t-1}\right) + g\Delta\theta + h\frac{\Delta\theta^2}{2}$$

- 要使得 $L\left(\theta^t\right)$ 极小，即让 $g\Delta\theta + h\frac{\Delta\theta^2}{2}$ 极小，可令：$\frac{\partial\left(g\Delta\theta + h\frac{\Delta\theta^2}{2}\right)}{\partial\Delta\theta} = 0$

求得 $\Delta\theta = -\frac{g}{h}$，故 $\theta^t = \theta^{t-1} + \Delta\theta = \theta^{t-1} - \frac{g}{h}$

参数 θ 推广到向量形式，迭代公式：$\theta^t = \theta^{t-1} - H^{-1}g$

这里 H 是 海森矩阵

从参数空间到函数空间

- GBDT 在函数空间中利用梯度下降法进行优化
- XGBoost 在函数空间中用牛顿法进行优化

注：实际上GBDT泛指所有梯度提升树算法，包括XGBoost，它也是GBDT的一种变种，这里为了区分它们，GBDT特指“Greedy Function Approximation：A Gradient Boosting Machine”里提出的算法，它只用了一阶导数信息。

从Gradient Descend 到 Gradient Boosting

参数空间

$$\theta^t = \theta^{t-1} + \theta_t$$

第t次迭代后的参数　第t-1次迭代后的参数　第t次迭代的参数增量

$$\theta_t = -\alpha_t g_t$$

参数更新方向为负梯度方向

$$\theta = \sum_{t=0}^{T} \theta_t$$

最终参数等于每次迭代的增量的累加和，

θ_0 为初值

从Gradient Descend 到 Gradient Boosting

参数空间

$$\theta^t = \theta^{t-1} + \theta_t$$

第t次迭代后的参数　第t-1次迭代后的参数　第t次迭代的参数增量

$$\theta_t = -\alpha_t g_t$$

参数更新方向为负梯度方向

$$\theta = \sum_{t=0}^{T} \theta_t$$

最终参数等于每次迭代的增量的累加和，

θ_0 为初值

函数空间

$$f^t(x) = f^{t-1}(x) + f_t(x)$$

第t次迭代后的函数　第t-1次迭代后的函数　第t次迭代的函数增量

$$f_t(x) = -\alpha_t g_t(x)$$

同样地，拟合负梯度

$$F(x) = \sum_{t=0}^{T} f_t(x)$$

最终函数等于每次迭代的增量的累加和，

$f_0(x)$ 为模型初始值，通常为常数

从Newton's Method 到 Newton Boosting

参数空间

$$\theta^t = \theta^{t-1} + \theta_t$$

第t次迭代后的参数　第t-1次迭代后的参数　第t次迭代的参数增量

$$\theta_t = -H_t^{-1} g_t$$

与梯度下降法唯一不同的就是参数增量

$$\theta = \sum_{t=0}^{T} \theta_t$$

最终参数等于每次迭代的增量的累加和，

θ_0 为初值

函数空间

$$f^t(x) = f^{t-1}(x) + f_t(x)$$

第t次迭代后的函数　第t-1次迭代后的函数　第t次迭代的函数增量

$$f_t(x) = -\frac{g_t(x)}{h_t(x)}$$

$$F(x) = \sum_{t=0}^{T} f_t(x)$$

最终函数等于每次迭代的增量的累加和，

$f_0(x)$ 为模型初始值，通常为常数

小结

- Boosting 算法是一种加法模型（additive training）

$$F(x)=\sum_{t=0}^{T} f_t(x)$$

小结

- Boosting 算法是一种加法模型（additive training）

$$F(x)=\sum_{t=0}^{T} f_t(x)$$

- 基分类器 f 常采用回归树[Friedman 1999] 和逻辑回归[Friedman 2000]。下文以回归树展开介绍。树模型有以下优缺点：

  - 可解释性强
  - 可处理混合类型特征
  - 具体伸缩不变性（不用归一化特征）
  - 有特征组合的作用
  - 可自然地处理缺失值
  - 对异常点鲁棒
  - 有特征选择作用
  - 可扩展性强，容易并行

  - 缺乏平滑性（回归预测时输出值只能输出有限的若干种数值）
  - 不适合处理高维稀疏数据

Gradient Boosting Tree 算法原理

- Friedman于论文” Greedy Function Approximation…”中最早提出GBDT

Gradient Boosting Tree 算法原理

- Friedman于论文” Greedy Function Approximation…”中最早提出GBDT

- 其模型 F 定义为加法模型：

$$F(x;w)=\sum_{t=0}^{T}\alpha_t h_t(x;w_t)=\sum_{t=0}^{T}f_t(x;w_t)$$

其中，x为输入样本，h为分类回归树，w是分类回归树的参数，α 是每棵树的权重。

Gradient Boosting Tree 算法原理

- Friedman于论文” Greedy Function Approximation…”中最早提出GBDT

- 其模型 F 定义为加法模型：

$$F(x;w)=\sum_{t=0}^{T}\alpha_t h_t(x;w_t)=\sum_{t=0}^{T}f_t(x;w_t)$$

其中，x为输入样本，h为分类回归树，w是分类回归树的参数，α 是每棵树的权重。

- 通过最小化损失函数求解最优模型：

$$F^*=\arg\min_F\sum_{i=0}^{N}L\left(y_i,F(x_i;w)\right)$$

NP难问题 -> 通过贪心法，迭代求局部最优解

Gradient Boosting Tree 算法原理

输入： $(x_i, y_i), T, L$

1. 初始化 f_0
2. for t = 1 to T do

2.1 计算响应：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1, 2, \ldots N$$

2.2 学习第t棵树：

$$w^* = \arg\min_w \sum_{i=1}^{N} \left(\tilde{y}_i - h_t(x_i; w)\right)^2$$

2.3 line search找步长：

$$\rho^* = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, F_{t-1}(x_i) + \rho h_t(x_i; w^*)\right)$$

2.4 令 $f_t = \rho^* h_t(x; w^*)$

更新模型：

$$F_t = F_{t-1} + f_t$$

3. 输出 F_T

Gradient Boosting Tree 算法原理

输入： $(x_i, y_i), T, L$

1. 初始化 f_0
2. for t = 1 to T do

2.1 计算响应：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1, 2, \ldots N$$

2.2 学习第t棵树：

$$w^* = \arg\min_w \sum_{i=1}^{N} \left(\tilde{y}_i - h_t(x_i; w)\right)^2$$

2.3 line search找步长：

$$\rho^* = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, F_{t-1}(x_i) + \rho h_t(x_i; w^*)\right)$$

2.4 令 $f_t = \rho^* h_t(x; w^*)$

更新模型：

$$F_t = F_{t-1} + f_t$$

3. 输出 F_T

Gradient Boosting Tree 算法原理

输入： $(x_i, y_i), T, L$

1. 初始化 f_0
2. for t = 1 to T do

2.1 计算响应：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1, 2, \ldots N$$

2.2 学习第t棵树：

$$w^* = \arg\min_w \sum_{i=1}^{N} \left(\tilde{y}_i - h_t(x_i; w)\right)^2$$

2.3 line search找步长：

$$\rho^* = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, F_{t-1}(x_i) + \rho h_t(x_i; w^*)\right)$$

2.4 令 $f_t = \rho^* h_t(x; w^*)$

更新模型：

$$F_t = F_{t-1} + f_t$$

3. 输出 F_T

Gradient Boosting Tree 算法原理

输入： $(x_i, y_i), T, L$

1. 初始化 f_0
2. for t = 1 to T do

2.1 计算响应：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1, 2, \ldots N$$

2.2 学习第t棵树：

$$w^* = \arg\min_w \sum_{i=1}^{N} \left(\tilde{y}_i - h_t(x_i; w)\right)^2$$

2.3 line search找步长：

$$\rho^* = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, F_{t-1}(x_i) + \rho h_t(x_i; w^*)\right)$$

2.4 令 $f_t = \rho^* h_t(x; w^*)$

更新模型：

$$F_t = F_{t-1} + f_t$$

3. 输出 F_T

Gradient Boosting Tree 算法原理

输入： $(x_i, y_i), T, L$

1. 初始化 f_0
2. for t = 1 to T do

2.1 计算响应：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1, 2, \ldots N$$

2.2 学习第t棵树：

$$w^* = \arg\min_w \sum_{i=1}^{N} \left(\tilde{y}_i - h_t(x_i; w)\right)^2$$

2.3 line search找步长：

$$\rho^* = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, F_{t-1}(x_i) + \rho h_t(x_i; w^*)\right)$$

2.4 令 $f_t = \rho^* h_t(x; w^*)$

更新模型：

$$F_t = F_{t-1} + f_t$$

3. 输出 F_T

Gradient Boosting Tree 算法原理

输入： $(x_i, y_i), T, L$

1. 初始化 f_0
2. for t = 1 to T do

2.1 计算响应：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1, 2, \ldots N$$

2.2 学习第t棵树：

$$w^* = \arg\min_{w} \sum_{i=1}^{N} \left(\tilde{y}_i - h_t(x_i; w)\right)^2$$

2.3 line search找步长：

$$\rho^* = \arg\min_{\rho} \sum_{i=1}^{N} L\left(y_i, F_{t-1}(x_i) + \rho h_t(x_i; w^*)\right)$$

2.4 令 $f_t = \rho^* h_t(x; w^*)$

更新模型：

$$F_t = F_{t-1} + f_t$$

3. 输出 F_T

Gradient Boosting Tree 算法原理

输入： $(x_i, y_i), T, L$

1. 初始化 f_0
2. for t = 1 to T do

2.1 计算响应：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1, 2, \ldots N$$

2.2 学习第t棵树：

$$w^* = \arg\min_w \sum_{i=1}^{N} \left(\tilde{y}_i - h_t(x_i; w)\right)^2$$

2.3 line search找步长：

$$\rho^* = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, F_{t-1}(x_i) + \rho h_t(x_i; w^*)\right)$$

2.4 令 $f_t = \rho^* h_t(x; w^*)$

更新模型：

$$F_t = F_{t-1} + f_t$$

3. 输出 F_T

Gradient Boosting Tree 算法原理

输入： $(x_i, y_i), T, L$

1. 初始化 f_0
2. for t = 1 to T do

2.1 计算响应：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1, 2, \ldots N$$

2.2 学习第t棵树：

$$w^* = \arg\min_w \sum_{i=1}^{N} \left(\tilde{y}_i - h_t(x_i; w)\right)^2$$

2.3 line search找步长：

$$\rho^* = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, F_{t-1}(x_i) + \rho h_t(x_i; w^*)\right)$$

2.4 令 $f_t = \rho^* h_t(x; w^*)$

更新模型：

$$F_t = F_{t-1} + f_t$$

3. 输出 F_T

Newton Boosting Tree 算法原理：详解 XGBoost

- 模型函数形式

- 目标函数
  - 正则项
  - 误差函数的二阶泰勒展开

- 回归树的学习策略
  - 打分函数
  - 树节点分裂方法
  - 缺失值的处理

- 其它特性

模型函数形式

给定数据集$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$，XGBoost进行additive training，学习K棵树，采用以下函数对样本进行预测：

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F},$$

模型函数形式

给定数据集 $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$，XGBoost进行additive training，学习K棵树，采用以下函数对样本进行预测：

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F},$$

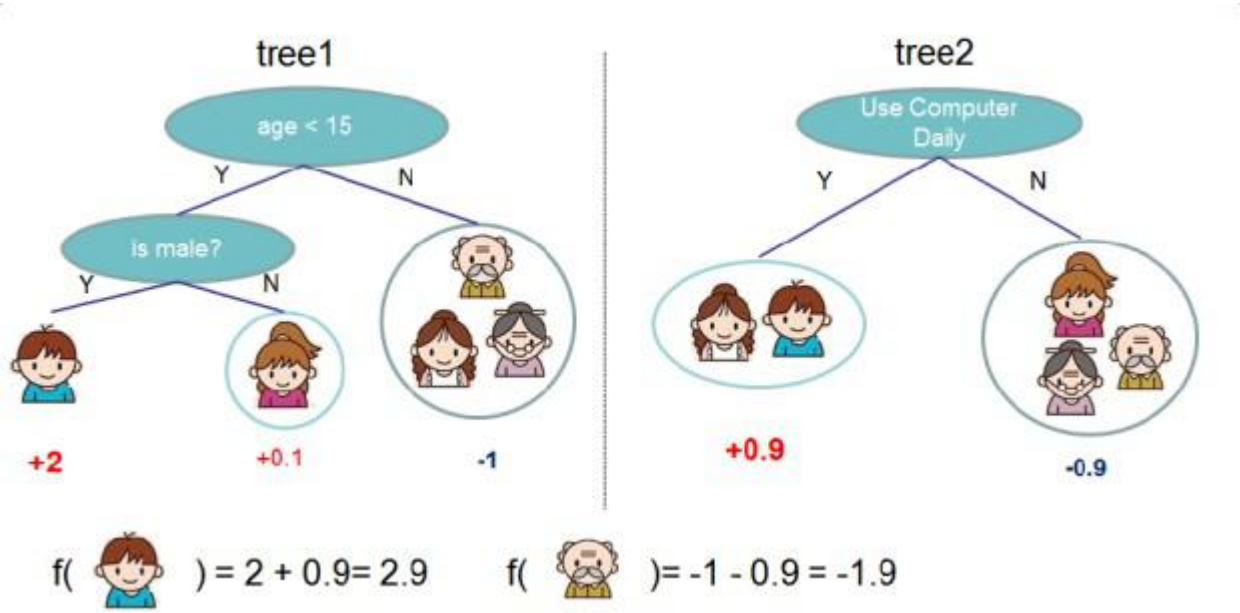
这里 $\mathcal{F}$ 是假设空间，f(x)是回归树（CART）：

$$\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}(q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T)$$

q(x)表示将样本x分到了某个叶子节点上，w是叶子节点的分数（leaf score），所以 $w_{q(x)}$ 表示回归树对样本的预测值

模型函数形式

- 例子：预测一个人是否喜欢电脑游戏



回归树的预测输出是实数分数，可以用于回归、分类、排序等任务中。对于回归问题，可以直接作为目标值，对于分类问题，需要映射成概率，比如采用逻辑函数 $\sigma(z)=\frac{1}{1+e^{-z}}$

目标函数

- 参数空间中的目标函数：

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

误差函数： 我们的模型有多拟合数据。

正则化项：惩罚复杂模型

误差函数可以是square loss，logloss等，正则项可以是L1正则，L2正则等。

Ridge Regression（岭回归）： $\sum_{i=1}^{n}\left(y_i-\theta^T x_i\right)^2+\lambda\|\theta\|^2$

LASSO： $\sum_{i=1}^{n}\left(y_i-\theta^T x_i\right)^2+\lambda\|\theta\|_1$

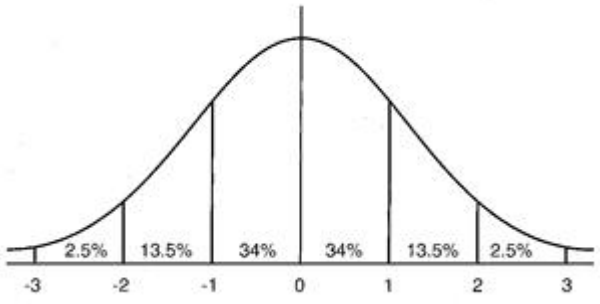
正则项

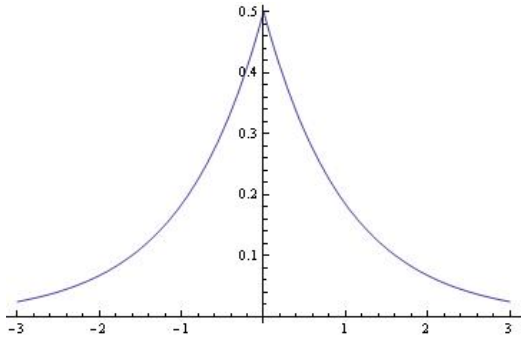
- 正则项的作用，可以从几个角度去解释：
  - 通过偏差方差分解去解释
  - PAC-learning泛化界解释
  - Bayes先验解释，把正则当成先验

正则项

- 正则项的作用，可以从几个角度去解释：
  - 通过偏差方差分解去解释
  - PAC-learning泛化界解释
  - Bayes先验解释，把正则当成先验

- 从Bayes角度来看，正则相当于对模型参数引入先验分布：





L2正则，模型参数服从高斯分布 $\theta \sim N(0,\sigma^2)$

对参数加了分布约束，大部分绝对值很小

L1正则，模型参数服从拉普拉斯分布

对参数加了分布约束，大部分取值为0

正则项

- XGBoost的目标函数（函数空间）

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

正则项对每棵回归树的复杂度进行了惩罚

正则项

- XGBoost的目标函数（函数空间）

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

正则项对每棵回归树的复杂度进行了惩罚

- 相比原始的GBDT，XGBoost的目标函数多了正则项，使得学习出来的模型更加不容易过拟合。

正则项

- XGBoost的目标函数（函数空间）

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

正则项对每棵回归树的复杂度进行了惩罚

- 相比原始的GBDT，XGBoost的目标函数多了正则项，使得学习出来的模型更加不容易过拟合。
- 有哪些指标可以衡量树的复杂度？

  树的深度，内部节点个数，叶子节点个数(T)，叶节点分数(w)...

  XGBoost采用的：

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

对叶子节点个数进行惩罚，相当于在训练过程中做了剪枝

误差函数的二阶泰勒展开

- 第t次迭代后，模型的预测等于前t-1次的模型预测加上第t棵树的预测：

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

误差函数的二阶泰勒展开

- 第t次迭代后，模型的预测等于前t-1次的模型预测加上第t棵树的预测：

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

- 此时目标函数可写作：

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i{}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

公式中 $y_i, \tilde{y}_i^{(t-1)}$ 都已知，模型要学习的只有第t棵树 f_t

误差函数的二阶泰勒展开

- 第t次迭代后，模型的预测等于前t-1次的模型预测加上第t棵树的预测：

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

- 此时目标函数可写作：

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

公式中 $y_i, \tilde{y}_i^{(t-1)}$ 都已知，模型要学习的只有第t棵树 f_t

- 将误差函数在 $\tilde{y}_i^{(t-1)}$ 处进行二阶泰勒展开：

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

公式中，$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

误差函数的二阶泰勒展开

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

- 将公式中的常数项去掉，得到:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

误差函数的二阶泰勒展开

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

- 将公式中的常数项去掉，得到:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

- 把 $f_t, \Omega(f_t)$ 写成树结构的形式，即把下式代入目标函数中

$$f(\mathbf{x}) = w_{q(\mathbf{x})} \qquad \Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$$

误差函数的二阶泰勒展开

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

- 将公式中的常数项去掉，得到：

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

- 把 $f_t, \Omega(f_t)$ 写成树结构的形式，即把下式代入目标函数中

$$f(\mathbf{x}) = w_{q(\mathbf{x})} \qquad \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

- 得到：

$$\tilde{L}^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

$$= \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2$$

误差函数的二阶泰勒展开

$$\tilde{L}^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

$$= \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2$$

对样本累加

对叶节点累加

- 怎么统一起来？

误差函数的二阶泰勒展开

$$\tilde{L}^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

$$= \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2$$

对样本累加　　对叶节点累加

- 怎么统一起来？

  定义每个叶节点j上的样本集合为 $I_j = \{i|q(x_i) = j\}$

  则目标函数可以写成按叶节点累加的形式：

$$\tilde{L}^{(t)} = \sum_{j=1}^{T}\left[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2\right] + \gamma T$$

$$= \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2\right] + \gamma T$$

误差函数的二阶泰勒展开

$$\tilde{L}^{(t)} = \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T$$

- 如果确定了树的结构（即q(x)确定），为了使目标函数最小，可以令其导数为0，解得每个叶节点的最优预测分数为：

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

代入目标函数，得到最小损失为：

$$\tilde{L}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

回归树的学习策略

- 当回归树的结构确定时，我们前面已经推导出其最优的叶节点分数以及对应的最小损失值，问题是怎么确定树的结构？

  暴力枚举所有可能的树结构，选择损失值最小的 - NP难问题

  贪心法，每次尝试分裂一个叶节点，计算分裂前后的增益，选择增益最大的

回归树的学习策略

- 当回归树的结构确定时，我们前面已经推导出其最优的叶节点分数以及对应的最小损失值，问题是怎么确定树的结构？

  暴力枚举所有可能的树结构，选择损失值最小的 - NP难问题

  贪心法，每次尝试分裂一个叶节点，计算分裂前后的增益，选择增益最大的

- **分裂前后的增益怎么计算？**

  ID3算法采用信息增益

  C4.5算法采用信息增益比

  CART采用Gini系数

  XGBoost呢？

XGBoost的打分函数

$$\tilde{L}^{*} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

标红部分衡量了每个叶子节点对总体损失的的贡献，我们希望损失越小越好，则标红部分的值越大越好。

XGBoost的打分函数

$$\tilde{L}^{*} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda}+\gamma T$$

标红部分衡量了每个叶子节点对总体损失的的贡献，我们希望损失越小越好，则标红部分的值越大越好。

因此，对一个叶子节点进行分裂，分裂前后的增益定义为：

$$Gain = \frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}-\gamma$$

Gain的值越大，分裂后 L 减小越多。所以当对一个叶节点分割时，计算所有候选(feature,value)对应的gain，选取gain最大的进行分割

树节点分裂方法（Split Finding）

- 精确算法

  遍历所有特征的所有可能的分割点，计算gain值，选取值最大的（feature，value）去分割

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*(I, *by* $\mathbf{x}_{jk}$) **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

树节点分裂方法（Split Finding）

- 近似算法

  对于每个特征，只考察分位点，减少计算复杂度

  - Global：学习每棵树前，提出候选切分点
  - Local：每次分裂前，重新提出候选切分点

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**

  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.

  Proposal can be done per tree (global), or per split(local).

end

for $k = 1$ ***to*** m **do**

  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$

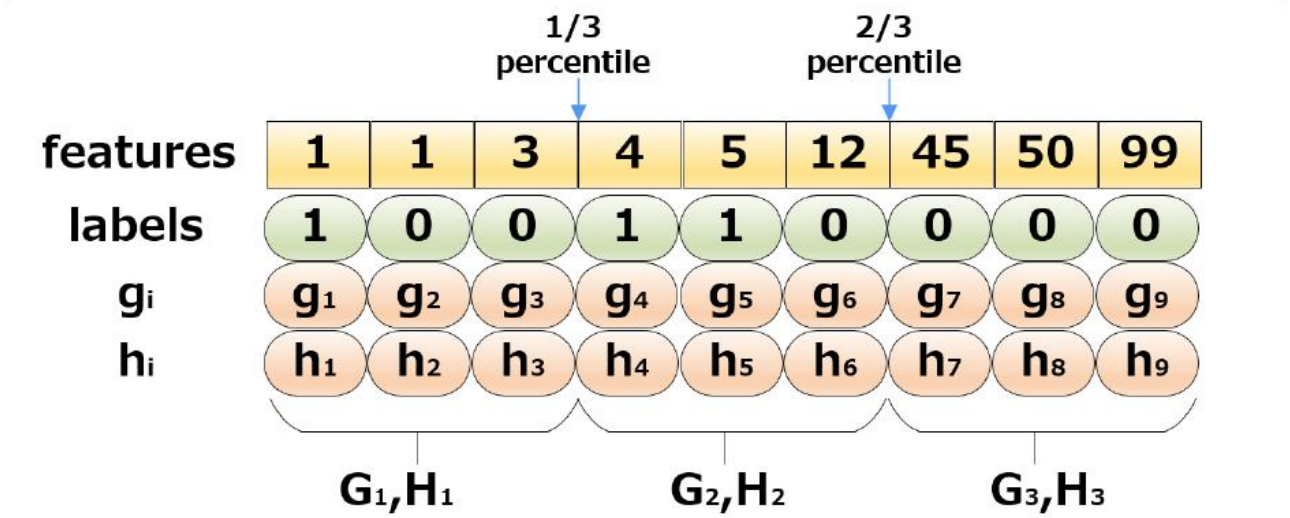
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$

end

Follow same step as in previous section to find max score only among proposed splits.

树节点分裂方法（Split Finding）

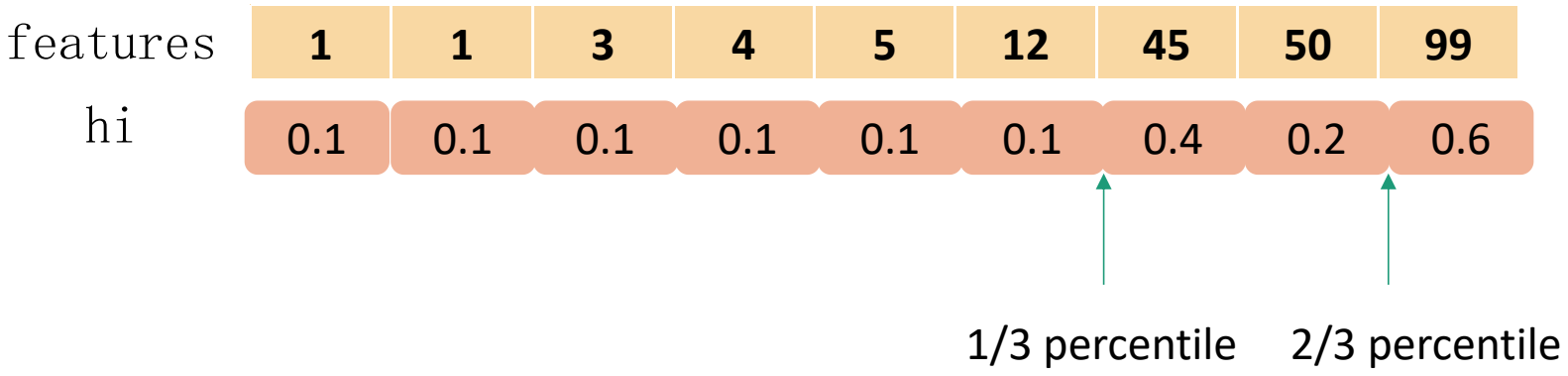
- 近似算法举例：三分位数



$$Gain = \max\{Gain, \frac{G_1^2}{H_1+\lambda} + \frac{G_{23}^2}{H_{23}+\lambda} - \frac{G_{123}^2}{H_{123}+\lambda} - \gamma, \frac{G_{12}^2}{H_{12}+\lambda} + \frac{G_3^2}{H_3+\lambda} - \frac{G_{123}^2}{H_{123}+\lambda} - \gamma\}$$

树节点分裂方法（Split Finding）

- 实际上XGBoost不是简单地按照样本个数进行分位，而是以二阶导数值作为权重(Weighted Quantile Sketch)，比如：



- 为什么用hi加权？

  把目标函数整理成以下形式，可以看出hi有对loss加权的作用

$$\sum_{i=1}^{n} \frac{1}{2} h_i (f_t(\mathbf{x}_i) - g_i/h_i)^2 + \Omega(f_t) + constant,$$

稀疏值处理

- 稀疏值：

缺失，类别one-hot编码，大量0值

- 当特征出现缺失值时，XGBoost

可以学习出默认的节点分裂方向

Algorithm 3: Sparsity-aware Split Finding

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, feature dimension
Also applies to the approximate setting, only collect statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I}, g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  // enumerate missing value goto right
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I_k,$ *ascent order by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
  // enumerate missing value goto left
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  for j *in sorted*$(I_k,$ *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default directions with max gain

XGBoost的其它特性

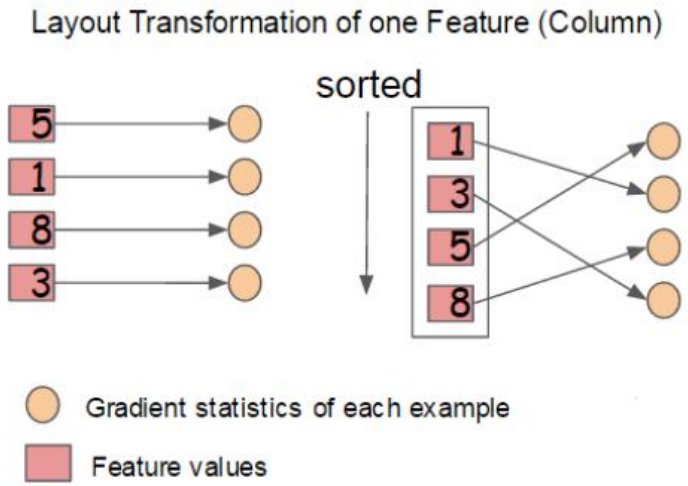
- 行抽样（row sample）
- 列抽样（column sample）
  借鉴随机森林
- Shrinkage（缩减），即学习速率
  将学习速率调小，迭代次数增多，有正则化作用
- 支持自定义损失函数（需二阶可导）

```
def logloss(preds, dtrain):
    labels = dtrain.get_label()
    # 将预测的score转化为概率
    preds = 1.0 / (1.0 + np.exp(-preds))
    grad = preds - labels
    hess = preds * (1.0 - preds)
    return grad, hess
```

XGBoost的系统设计

- **Column Block**
  - 特征预排序，以column block的结构存于内存中
  - 存储样本索引（instance indices）
  - block中的数据以稀疏格式（CSC）存储

这个结构加速了split finding的过程，只需要在建树前排序一次，后面节点分裂时直接根据索引得到梯度信息



- **Cache Aware Access**
  - column block按特征大小顺序存储，相应的样本的梯度信息是分散的，造成内存的不连续访问，降低CPU cache 命中率
  - 缓存优化方法
    - 预取数据到buffer中（非连续->连续），再统计梯度信息
    - 调节块的大小

更高效的工具包 LightGBM

- 速度更快

Data	xgboost	xgboost_hist	LightGBM
Higgs	3794.34 s	551.898 s	238.505513 s
Yahoo LTR	674.322 s	265.302 s	150.18644 s
MS LTR	1251.27 s	385.201 s	215.320316 s
Expo	1607.35 s	588.253 s	138.504179 s
Allstate	2867.22 s	1355.71 s	348.084475 s

- 内存占用更低

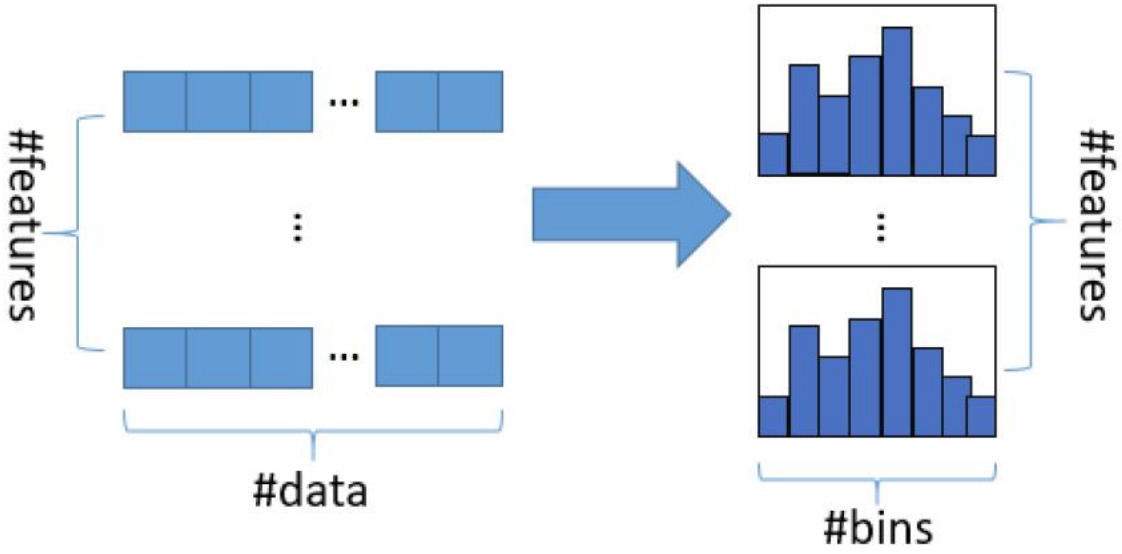
Data	xgboost	xgboost_hist	LightGBM
Higgs	4.853GB	3.784GB	**0.868GB**
Yahoo LTR	1.907GB	1.468GB	**0.831GB**
MS LTR	5.469GB	3.654GB	**0.886GB**
Expo	1.553GB	1.393GB	**0.543GB**
Allstate	6.237GB	4.990GB	**1.027GB**

- 准确率更高（优势不明显，与XGBoost相当）

LightGBM的改进

- 直方图算法

把连续的浮点特征值离散化成k个整数，同时构造一个宽度为k的直方图。在遍历数据的时候，根据离散化后的值作为索引在直方图中累积统计量，当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点

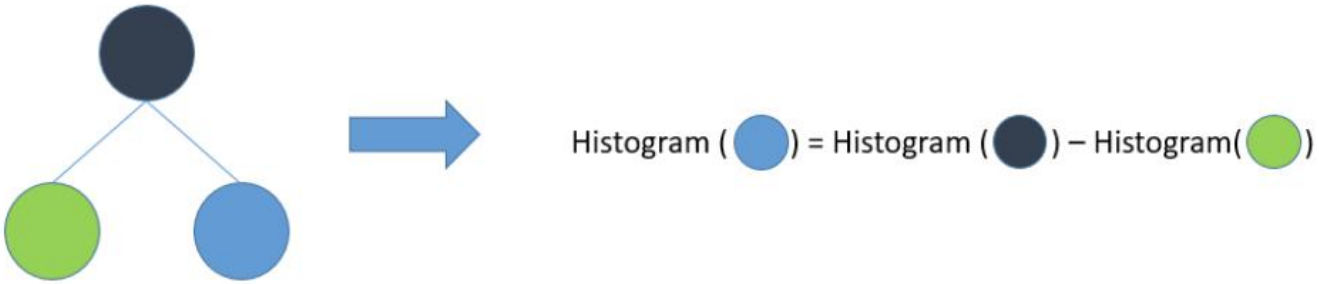


- 减小内存占用，比如离散为256个bin时，只需要8bit，节省7/8
- 减小了split finding时计算增益的计算量， 从O(#data) 降到O(#bins)

LightGBM的改进

- 直方图差加速

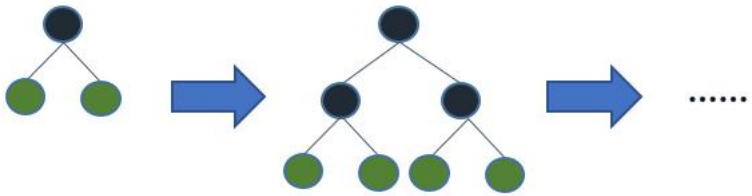
一个叶子的直方图可以由它的父亲节点的直方图与它兄弟节点的直方图做差得到，提升一倍速度



思考：选取哪个子节点统计直方图？

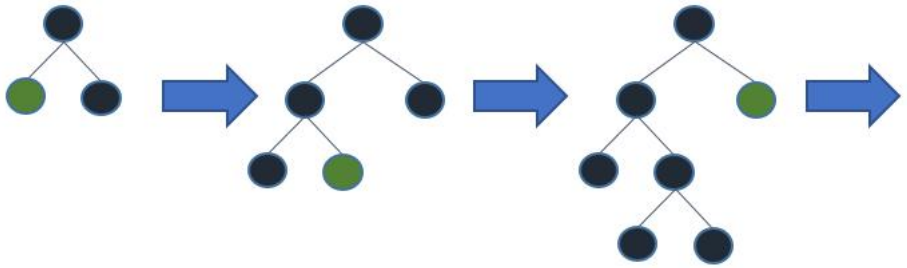
LightGBM的改进

- 建树过程的两种方法：Level-wise和Leaf-wise



Level-wise tree growth

XGBoost
同一层所有节点都做分裂，最后剪枝

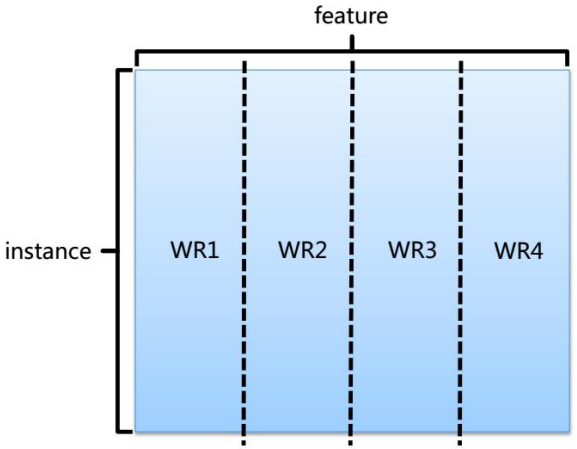


Leaf-wise tree growth

LightGBM
选取具有最大增益的节点分裂
容易过拟合，通过max_depth限制

LightGBM的改进

- 并行优化（**Optimization in parallel learning**）
  - 传统的特征并行
    1. 垂直切分数据，每个worker只有部分特征
    2. 每个worker找到局部最佳切分点（feature，threshold）
    3. worker之间互相通信，找到全局最佳切分点
    4. 具有全局最佳切分特征的worker进行节点分裂，然后广播切分后左右子树的instance indices
    5. 其他worker根据广播的instance indices进行节点分裂



特征并行示意图

缺点：

- split finding计算复杂度O(#data)，当数据量大时会比较慢
- 网络通信代价大，需要广播instance indices

LightGBM的改进

- 并行优化（**Optimization in parallel learning**）
  - LightGBM的特征并行

    每个worker保存所有数据集
    1. 每个worker在其特征子集上寻找最佳切分点
    2. worker之间互相通信，找到全局最佳切分点
    3. 每个worker根据全局最佳切分点进行节点分裂

    优点：

    避免广播instance indices，减小网络通信量

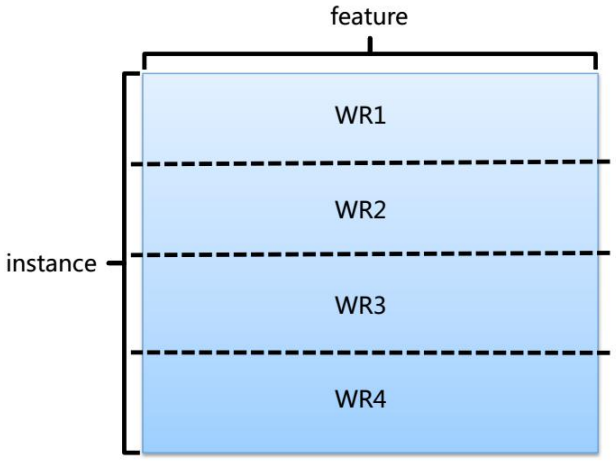
    缺点：

    split finding计算复杂度没有减小

    且当数据量比较大时，单个worker存储所有数据代价高

LightGBM的改进

- 并行优化（**Optimization in parallel learning**）
  - 传统的数据并行
    1. 水平切分数据，每个worker只有部分数据
    2. 每个worker根据本地数据统计局部直方图
    3. 合并所有局部直方图得到全局直方图
    4. 根据全局直方图进行节点分裂



数据并行示意图

缺点：网络通信代价巨大

采用point-to-point communication algorithm，
每个worker通信量 O(#machine * #feature * #bin)

采用collective communication algorithm，
每个worker通信量O(2 * #feature * #bin)

LightGBM的改进

- 并行优化（**Optimization in parallel learning**）
  - LightGBM的数据并行
    - 不同的worker合并不同特征的局部直方图
    - 采用直方图做差算法，只需要通信一个节点的直方图

    通信量减小到 O(0.5 * #feature* #bin)

    Voting parallel，参考论文“A Communication-Efficient Parallel Algorithm for Decision Tree”

LightGBM的改进

- Gradient-based One Side Sampling (GOSS)

  在每一次迭代前，利用了GBDT中的样本梯度和误差的关系，对训练样本进行采样: 对误差大（梯度绝对值大）的数据保留；对误差小的数据采样一个子集，但给这个子集的数据一个权重，让这个子集可以近似到误差小的数据的全集。这么采样出来的数据，既不损失误差大的样本，又在减少训练数据的同时不改变数据的分布，从而实现了在几乎不影响精度的情况下加速了训练。

- Exclusive Feature Bundling (EFB)

  在特征维度很大的数据上，特征空间一般是稀疏的。利用这个特征，我们可以无损地降低GBDT算法中需要遍历的特征数量，更确切地说，是降低构造特征直方图（训练GBDT的主要时间消耗）需要遍历的特征数量。

https://www.zhihu.com/question/64943934

参考文献

- Greedy function approximation a gradient boosting machine. *J.H. Friedman(1999).*
- *Additive logistic regression a statistical view of boosting. Friedman(2000).*
- XGBoost: A Scalable Tree Boosting System. T. Chen, C. Guestrin (2016).
- A Highly Efficient Gradient Boosting Decision Tree. Guolin Ke (2017).
- Introduction to Boosted Trees. T. Chen
- Tree Boosting With XGBoost. Didrik Nielsen
- 《统计学习方法》李航. 附录梯度下降法与牛顿法

- https://github.com/Microsoft/LightGBM/wiki/Features

- XGBoost 与 Boosted Tree

- GBDT详解，火光摇曳

- 泰勒公式，维基百科

谢谢聆听！